On the Efficiency of Sharing Distilled Data in Federated Learning

Pascal Epple
Department of Computer Science, EPF Lausanne, Switzerland

Figure 1. Example of distilled images generated from a subset of 50 base images of CIFAR-10 using the distillation technique presented by Cazenavette et al. [1]. Training a standard CNN using these distilled images yields a trained model capable of test accuracy significantly better than the same model trained on the base images. (Can you guess which classes these distilled images represent? From top to bottom: Airplane, Automobile, Deer, Horse, Truck)

***Abstract*—In this semester project, we explore the efficiency of sharing distilled data in the context of Federated Learning (FL). We assess the local performance of a distillation technique based on matching trajectories and evaluate a global model trained on the aggregated distilled dataset. Our results demonstrate that training a model on the aggregated synthetic data leads to significantly better performance than a model trained on the base, non-distilled subset of data. During the generation of synthetic data, we face challenges related to ill training behavior that are difficult to address due to the high computational cost of the distillation technique. Despite these limitations, our approach shows potential for reducing communication cost and for enhancing data privacy in FL.**

I. Introduction

Federated Learning has emerged as a promising approach for collaborative learning in large-scale distributed systems with a huge number of networked clients. Due to the limited bandwidth between clients [2], previous research attempts to improve communication efficiency and to speed up convergence. This communication bottleneck becomes even more important today as modern neural network have over hundreds of million parameters, and thereby hinders the large-scale deployment for federated learning models.

What alternative to sending model weights exist in order to share knowledge with other clients? Motivated by this question, we go back to the essence of what neural networks learn from: data. However, due to privacy reasons, we cannot simply send raw data to other clients. This is where dataset distillation comes into play. Dataset distillation reduces the train dataset to a much smaller synthetic dataset such that a model trained on it has similar performance to a model trained on the whole, real data. By sharing distilled data instead of model weights directly, we address both the communication cost and data privacy concerns inherent in federated learning. Indeed, sharing way less data opens the door to encrypted computation and secure communication between clients.

In this work, we examine the efficiency of sharing distilled data within the context of Federated Learning. To

accomplish this, we first examine the local performance of a distillation technique proposed by Cazenavette et al. [1]. Then, we compare the performance of a global model trained on the distilled datasets generated by each client to a model collaboratively trained via a classical federated averaging approach.

II. Background and Related Work

Federated Learning. Federated Learning was first introduced by McMahan et al. [3], where models can be learned collaboratively from decentralized data through model exchange between clients and a central server. The proposed federated learning scheme *FedAvg* [3] updates the global model by averaging the parameters of the received models. In comparison to distributed learning, federated learning addresses more practical challenges such as communication efficiency [4], data heterogeneity [5] and privacy protection [6].

In a federated learning scenario, each client k trains a machine learning models with weights w_k on a local datasets $\mathcal{D}_k = \{(x_i)|i = 1, 2, ..., n_k\}$, where x_i refers to a data point with its associated label y_i. The number of local data points n_k can be different across clients and the distribution can also be Non-IID. The goal of client k is to locally train a machine learning model in order to minimize

$$F_k(w) = \frac{1}{n_k} \sum_{i=1}^{n_k} f_i(w) \tag{1}$$

where $f_i(w)$ is the loss function on one data point x_i with corresponding label y_i. The final goal is to minimize the aggregated local goals $F_k(w)$ that were computed in (1):

$$f(w) = \sum_{k=1}^{m} \frac{n_k}{n} F_k(w) \tag{2}$$

Most federated learning techniques exchange the weights of the model or gradients for learning updates. This may cause an increase in communication cost as model size increases. This bottleneck can make it difficult for clients to frequently upload the newly trained models or the gradients to the server.

Dataset Distillation. Dataset distillation was first introduced by Wang et al. [7]. The authors expressed the parameters of the model as a function of the images to be distilled and optimized them using gradient-based hyperparameter optimization [8]. The goal of the method is to extract knowledge from the complete dataset and to derive a much smaller synthetic dataset such that a model trained on it can have comparable performance with a model trained on the real dataset. Figure 2 illustrates data distillation.

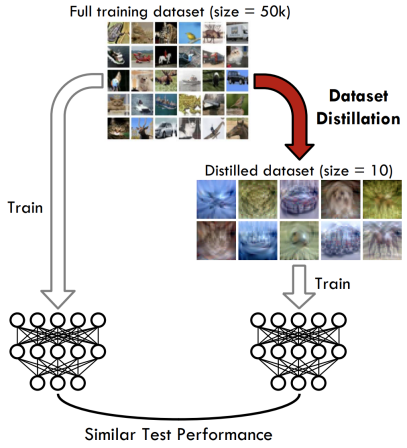

Figure 2. Dataset distillation aims to generate a small synthetic dataset for which a model trained on it can achieve a similar test performance as a model trained on whole, complete train set. Figure and caption taken from Cazenavette et al. [1]

There are multiple approaches to data distillation. Outstanding results have been achieved by methods based on matching outputs or gradients [1], [9], [10]. Nguyen et al. [11] perform Kernel Inducing Points (KIP) and Label Solve (LS) to find the optimal synthetic dataset. We refer the reader to the work of Yu et. al [12] for more detailed descriptions of the different dataset distillation methods that are in use.

Dataset Distillation in Federated Learning. Some previous works in dataset distillation, such as Zhao et al. [9], have mentioned the potential benefit of dataset distillation in a federated setting. However, they did not pursue a detailed analysis on the matter. In recent research, Song et al. [13] claim to be "the first to introduce a decentralized dataset distillation scheme in a federated learning systems, where distilled data instead of models are uploaded to the server". They considered distillation methods that are coreset-based [14] and KIP-based [11]. Their method enhances more robust training performance on Non-IID data and significantly reduces communication costs.

III. Method

To explore the effectiveness of dataset distillation in a federated scenario, we make use of DecentralizePy [15] to set up a joint learning task with k participating clients, to whom we refer to as *nodes*.

Every node owns a local dataset $\mathcal{D}_k$ of size n_k and distills it in into a much smaller dataset $\mathcal{S}_k$ of size m_k, referred to as the local synthetic dataset. The end goal of the local distillation process is to ensure that a model trained on S_k matches the performance of a model trained using the complete local dataset $\mathcal{D}_k$.

First, we discuss in section III-A the local data distillation method employed at the node level. Then, in section III-B,

we detail the data aggregation and training procedure done by the server.

A. Dataset Distillation by Matching Training Trajectories

We use the dataset distillation by matching training trajectories presented in Cazenavette et al. [1]. This technique produces outstanding synthetic datasets and significantly outperforms modern techniques such as KIP and LS. As described by the authors, this method explicitly encourages the distilled dataset to induce similar long-range network parameter trajectories as the real dataset, resulting in a synthetically-trained network that performs similarly to a network trained on real data. The distillation method can be boiled down into two separate parts. The first one is responsible for generating expert trajectories and is presented in section III-A1. The second part consists of the actual dataset distillation phase of the algorithm. Cazenavette el al. [1] refer to it as the long-range parameter matching phase and it is discussed in section III-A2. Algorithm 1 illustrates the complete local distillation process.

1) Expert Trajectories: As described in Cazenavette et al. [1], the core of the method involves using *expert trajectories* τ^* to guide the distillation of the synthetic dataset. Expert trajectories represent the temporal progression of parameters $\{\theta_t^*\}_0^T$ stored during the training of a neural network using the *complete, real local dataset*. To generate these expert trajectories, a straightforward approach is used: a large number of networks are trained on the real dataset and the parameters after every training epoch are saved. The resulting sequences of parameters are named "expert trajectories" due to the fact that they represent a theoretical upper bound for the network trained on the synthetic data. The latter is defined as the student network, whose snapshot parameters $\hat{\theta}_t$ at training step t are referred to as student parameters. During the distillation process, the goal is to shape the synthetic dataset in such a way that it will induce a similar trajectory to the parameters of the student network as the one taken by the expert parameters, given that the student network is initialized with the same parameters as the expert at a particular epoch. Notice that generating these expert trajectories is a complete orthogonal task to the actual data distillation. The trajectories can hence be pre-computed before the actual distillation phase.

2) Long-Range Parameter Matching: After generating the expert trajectories $\{\theta_t^*\}_0^T$, we now present how the distillation process learns from the sequences of parameters.

At each distillation step, the student parameters $\hat{\theta}_t$ are initialized to a random timestep θ_t^* of one of the previously generated expert trajectories. As the parameters of the expert models more change that much in the late stages of training, an upper bound T^+ on t is defined. This lets us ignore the less informative later parts of the expert trajectories. With the student network initialized, the parameters of the student network get updated for N gradient descent steps with respect to the classification loss of the synthetic data:

$$\hat{\theta}_{t+n+1} = \hat{\theta}_{t+n} - \alpha_k \nabla l(\mathcal{A}(\mathcal{S}_k; \hat{\theta}_{t+n}), \tag{3}$$

where $\mathcal{A}$ denotes some differentiable augmentation technique [9], and α_k is the (trainable) learning rate of the student network of each node. These same differentiable augmentation techniques are also used in order to augment the data while generating the expert trajectories. We put emphasis on the fact that the augmentation techniques need to be differentiable as this ensures the ability to back-propagate gradients up until the input that gets distilled. There are numerous examples of such differentiable augmentation techniques for images: random cropping, random rotating, adding Gaussian noise, etc. From this point, the (previously generated) expert parameters θ_{t+M}^* are retrieved. They are the parameters of the expert model M training updates after the ones used to initialize the student network. Finally, the distilled images are updated according to the weight matching loss which is defined as the normalized squared L_2 error between the updated student parameters $\hat{\theta}_{t+N}^*$ and the known future expert parameters θ_{t+M}^*:

$$\mathcal{L} = \frac{\|\hat{\theta}_{t+N} - \theta_{t+M}^*\|_2^2}{\|\hat{\theta}_t^* - \theta_{t+M}^*\|_2^2}$$

Normalization is done by the distance traveled by the expert, such that there will still be a strong signal when taking expert parameters from later stages of training which tend to be closer to one another. This loss function is then minimized and the pixels of the distilled dataset, along with the trainable learning, get updated by back-propagating the gradient through all N updates that were done on the student network.

Algorithm 1 Dataset Distillation via Trajectory Matching

Input: $\{\tau_i^*\}$: set of expert parameter trajectories trained on $\mathcal{D}_k$
Input: M: # of updates between starting and target expert params.
Input: N: # of updates to student network per distillation step
Input: $\mathcal{A}$: Differentiable augmentation function
Input: $T^+ < T$: Maximum start epoch
Initialize distilled data $\mathcal{S}_k \sim \mathcal{D}_k$
Initialize trainable learning rate $\alpha_k := \alpha_{0,k}$
for each distillation step... **do**
    ▷ Sample expert trajectory with a lot of params
    ▷ Choose random start epoch
    ▷ Initialize student network with expert params:
        $\hat{\theta} := \theta_t^*$
    for $n = 0 \rightarrow N - 1$ **do**
        ▷ Sample a mini-batch of distilled images:
            $b_{t+n} \sim \mathcal{S}_k$
        ▷ Update student network w.r.t. classification loss:
            $\hat{\theta}_{t+n+1} = \hat{\theta}_{t+n} - \alpha_k \nabla l(\mathcal{A}(b_{t+n}; \hat{\theta}_{t+n}))$
    end for
    ▷ Compute loss between ending student:
        $\mathcal{L} = \|\hat{\theta}_{t+N} - \theta_{t+M}^*\|_2^2 / \|\hat{\theta}_t^* - \theta_{t+M}^*\|_2^2$
    ▷ Update $\mathcal{S}_k$ and α_k with respect to $\mathcal{L}$
end for
Output: distilled data $\mathcal{S}_k$ and learning rate α_k

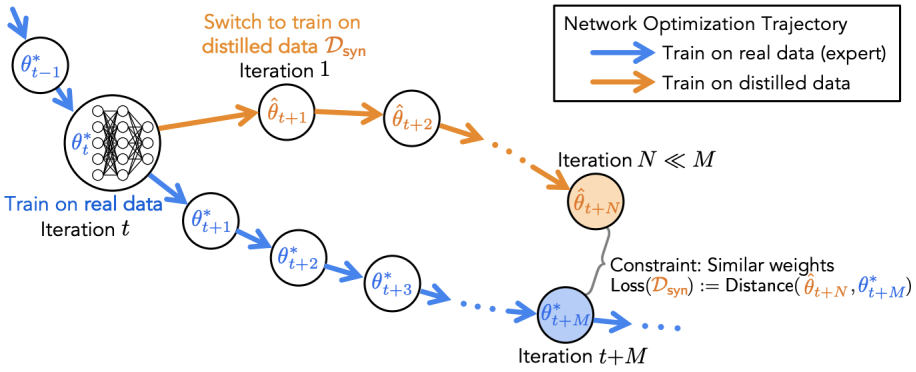

Figure 3. Illustration of long-range parameter matching between training on real data and training on distilled data as done by Cazenavette et al. [1]. Given the same starting parameters sampled from one expert trajectory, the distilled data $\mathcal{D}_{\text{syn}}$ is trained such that N on it matches the same result (in parameter space) from much more M steps on the real data. Figure and caption are taken from [1].

B. Data aggregation and global model training

In federated learning, the goal is to learn a collaboratively learn a global model. We adapt the setting such that data instead of model weights are shared.

Each node k sends the synthetic dataset $\mathcal{S}_k$ that it generated from its own local dataset $\mathcal{D}_k$ to the central server. This central server will simply combine and stack the distilled datasets into one single dataset $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cdots \cup \mathcal{S}_m$. The resulting synthetic dataset is then used to train a global model from scratch.

Notice that the nodes do not send the learned learning rate α_k to the server. Instead, the server independently conducts a hyper-parameter search to identify the optimal learning rate and optimizer for the global model.

IV. Experiments

In section IV-A, we focus on the performance of the distillation technique presented by Cazenavette et al. [1] in a decentralized setting. Then, in section IV-B, we study the efficiency of the global network trained on the aggregated distilled datasets that were generated by every single node.

Experimental Settings. Our experiments are conducted using the DecentralizePy framework. We consider a setting of 1 server and 10 participating nodes where each node initially receives an equal share (i.e. 10% of the trainset) of the complete original train data. The local datasets are split into a train set and a validation set. By setting the *number of shards*, one can decide on the the number of classes that are sent to every node. The partitioning of the data is done using the already implemented *KShardPartitioner*, which splits the data on each node in an IID-way. Consequently, every node will roughly have the same number of data points for each class it received for the training and the validation set. The testset of the complete dataset is kept on the server and serves the later purpose of evaluating the global model trained on the aggregated distilled datasets of each node. As no communication is needed between the different nodes during distillation, the structure of the graph connecting the nodes is not important. What simply matters is that every node has a direct communication channel with the central server. Each node will use this connection to keep the server updated about its distillation progress and to send the final best performing set of distilled images it generated.

We evaluate our method on CIFAR-10 [16].

A. Efficiency of Data Distillation on Single Nodes

On every node, we want to employ the distillation technique described in Algorithm 1 to generate the distilled images. However, Cazenavette et al. [1] do not mention any evaluation of their technique on subsets of CIFAR-10. It is thus crucial to ensure that each node is able to properly distill the local dataset it has been given.

If not mentioned otherwise, the hyper-parameters used for each setting are the same as Cazenavette et al. [1] used to distill the whole, complete CIFAR-10 trainset (available here. We also use the same differentiable augmentations as proposed by Cazenavette et al. [1]. We do not employ ZCA-whitening. For each setting, we pre-compute 30 expert trajectories.

Baseline. As a reminder, the distillation task of each node k is to generate a distilled data set $\mathcal{S}_k$ from a much larger data set $\mathcal{D}_k$. As presented in Algorithm 1, the distilled images are initially sampled from $\mathcal{D}_k$. We refer to this small subset of images as the *base* images $\mathcal{B}_k$ of node k. To assess the efficiency of the distillation technique on each node k, we train randomly initialized neural networks from scratch on the distilled data $\mathcal{S}_k$ and on the corresponding base images $\mathcal{B}_k$. We then evaluate the performance of the trained models on the validation set of each node.

Network architecture. We consistently employ a simple ConvNet architecture designed by Gidaris and Komodakis [17]. The architecture consists of several convolutional blocks, each made out of a 3×3 convolution layer with 128 filters, Instance normalization [18] , RELU, and 2×2

average pooling with a stride of 2. After the convolutional blocks, the logits are produced by one single linear layer.

1) Distillation for Different Shards: We focus on the performance of the distillation technique for different numbers of shards. We vary the number of shards between 2, 5 and 10. Using this approach, we can assess the method's robustness to data heterogeneity. For each selected number of shards, we decide on a number of images per class that every node distills. As an example, given a setup where the dataset is partitioned with 2 shards and the number of images per class equals 10, each node distills 20 images. The number of images per class that each node distills is either 1 or 10 (except for 10 shards, where we only set it to 1). These numbers are chosen such that that the final distilled dataset is always much smaller than the original, complete local dataset. In the most extreme scenario (10 images per class and 5 shards), the distilled dataset size is still only 1% the size of the original, complete local dataset.

In every setting, we run the distillation for 2000 iterations (with our computational resources, this is the best possible). Every 200 iterations, we compute the mean accuracy of three randomly initialized neural network trained from scratch on the distilled images. The final synthetic dataset consists of the images that led to the best performance during this evaluation phase. We present and discuss the computed results for each setting.

To save time and resources, for a fixed number of shards and varying number of images per class, we distill on the same sets of expert trajectory.

2 Shards. The distilled images generated by the nodes significantly outperform the baseline. The update parameter of the learnable learning rate is set to $1e-7$.

For 1 image per class, the mean improvement on the respective validation accuracy is of 16.49%. When 10 images per class are distilled, it slightly reduces to 13.29%.

In Figure 4, we plot the results of the distillation process on the node with rank 3. On this node, we notice that training on 2 distilled images outperforms training on 10 non-distilled images. This phenomenon is not unique and is in fact observed on a total of 7 nodes.

We notice that the validation accuracy increases for all except one node when increasing the number of distilled images per class. Interestingly, for the node of rank 4, there is a drop of 2.5% in validation accuracy when training on 10 times the amount of synthetic data. This unexpected result is discussed in section V. Excluding this particular node, the mean improvement equals 11.75%. When including the ill-behaving node, the mean improvement decreases to 10.31%.

5 Shards. On each node, we notice that distilled images always lead to significantly better performance than their non-distilled counterparts. Hence, the distillation technique is working efficiently. The update parameter of the learnable learning rate is set to $1e-6$.

When 1 image per class is distilled, we observe a mean

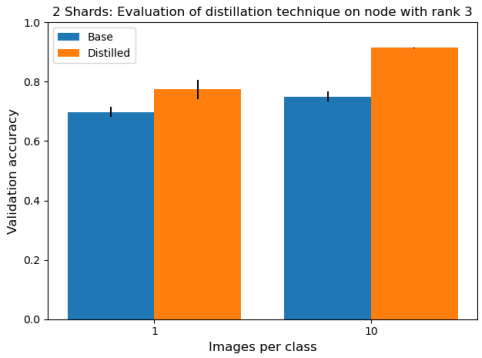

Figure 4. Effectiveness of the distillation technique with 2 shards on node of rank 3. For 1 image per class, the validation accuracy improves from 69.80% to 77.43%. For 10 images per class, we notice an increase from 75.03% to 91.40%. It is remarkable that 2 distilled images outperform 20 non-distilled images by 2.40%. Standard errors are computed by training 3 neural networks from scratch for every setting.

increase of 21.6% in validation accuracy with respect to the performance on the 5 base images. We also notice a mean increase of 20.38% when 10 images per class are distilled.

Figure 5 showcases the results of the distillation process on the node with rank 0. Similarly to what we previously noticed for 2 shards, we see that 5 distilled images slightly outperform 50 non-distilled images. This remarkable phenomenon actually happens on all except one node.

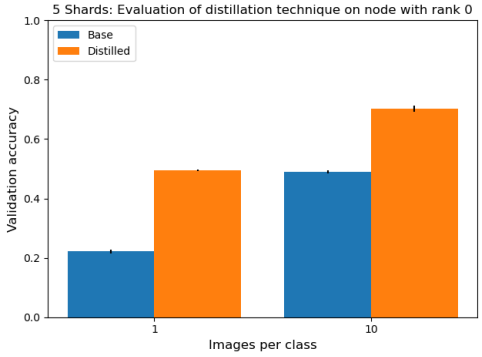

Figure 5. Effectiveness of the distillation technique with 5 shards on node of rank 0. **For every setting, 3 neural networks are trained from scratch.** For 1 image per class, the validation accuracy improves from 22.13% to 49.53%. For 10 images per class, we notice an increase from 49.00% to 70.03%. We also notice that 5 distilled images slightly outperform 50 base images, namely by 0.53%. Standard errors are computed by training 3 neural networks from scratch for every setting.

When increasing the number of distilled images per class from 1 to 10, we notice a mean increase in performance of 15.12%.

10 Shards. The mean improvement on the nodes' respective validation accuracy is of 16.40% and the distilled images generated by the nodes always significantly outperform the baseline. The update parameter of the learnable learning rate is set to $1e-6$.

Table I
THE PREDICTION ACCURACY COMPARSON BETWEEN OUR FL SCHEME BASED ON DATASET DISTILLATION AND FEDAVG ON CIFAR-10 DISTRIBUTED IN 10 NODES

Shards	1 IPC Base	1 IPC Distilled	10 IPC Base	10 IPC Distilled	FedAvg
2	16.54 ± 0.21%	28.43 ± 0.32%	40.06 ± 0.27 %	44.57 ± 0.38%	**56.62 ± 0.42%**
5	21.57 ± 0.14%	40.95 ± 0.25%	45.60 ± 0.33 %	53.43± 0.19 %	**66.26 ± 0.49%**
10	31.36 ± 0.26 %	37.61 ± 0.16 %	-	-	**69.08 ± 0.38%**

We run FedAvg 3 times for 20 iterations and the number of rounds is set to 2. For the other settings, we train three randomly initialized models from scratch on the aggregated datasets.

In Figure 6, we present the evaluation of the distillation distilled images for node with rank 9.

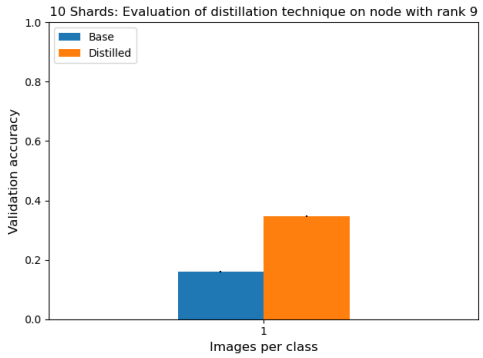

Figure 6. Effectiveness of the distillation technique with 10 shards on node of rank 9. The validation accuracy improves from 16.07% to 34.63% when distilling 1 image per class. Standard errors are computed by training 3 neural networks from scratch for every setting.

Resources and Training Time. We use Google Colab [19] which is a cloud-based Jupyter notebook [20] environment. It offers free but not unlimited access to GPUs to users with a Google account. We sequentially distill the local dataset of every node for every setting presented in section IV-A. The computations are done with a Tesla T4. The local training time for every setting is reported in table II.

Table II
LOCAL TRAINING TIME OF THE DISTILLATION METHOD

Shards	IPC	2k Iterations (in min.)
2	1	22
	10	24
5	1	18
	10	32
10	1	25
	10	-

B. Evaluation on Aggregated Distilled Datasets

We explore the performance of one global model trained on the aggregated distilled datasets such as explained in III-B. This model has exactly the same architecture as described in section IV-A. The data augmentation technique is identical to the one used on the nodes.

In Algorithm 1, the learning rate α_k of the student network is optimized along the distillation process to capture the full potential of the network. Finding the optimal hyper-parameters is thus an extremely important task to assess the performance of the aggregated distillated images. To ease this hyper-parameter search, we use the open source framework Optuna [21]. Optuna is a python library that enables automatic fine-tuning of neural networks. In addition to the learning rate, we also decide to determine the best possible optimizer between SGD [22] and Adam [23].

We compare the performance of our method to a model trained on the base images of the distilled data and to *FedAvg*. The results of the best performing models are summarized in Table IV-A1.

Looking at Table IV-A1, we observe that across all the settings we considered, a model trained on the aggregated distilled dataset consistently outperforms a model trained on the aggregated base dataset.

Furthermore, the local effect we observed in section IV-A, where one single distilled image outperformed 10 base images, disappears when aggregating the datasets. Additionally, we notice that the setting with 5 shards outperforms the setting with 10 shards when distilling 1 image per class. We have two possible interpretations for this disparity. First, it could be attributed to a trade-off between the number of data samples used for distillation and the number of classes on a node. Indeed, the distilled images in the setting of 5 shards are a condensation of a larger number of samples per class than in the setting of 10 shards. One other interpretation is discussed in section V and concerns the detection of ill-behaving nodes during training. In all cases, based on the results, our method seems to be robust to data heterogeneity among nodes, which aligns with the findings of [13].

Lastly, it is worth to mention that the results we obtain are still far off a more traditional federated learning technique such as FedAvg. However, FedAvg comes with a high cost of communication between the nodes, which for some distributed settings might become a serious bottleneck. In contrast, our method comes with a great advantage as it strongly reduces the communication cost between the clients

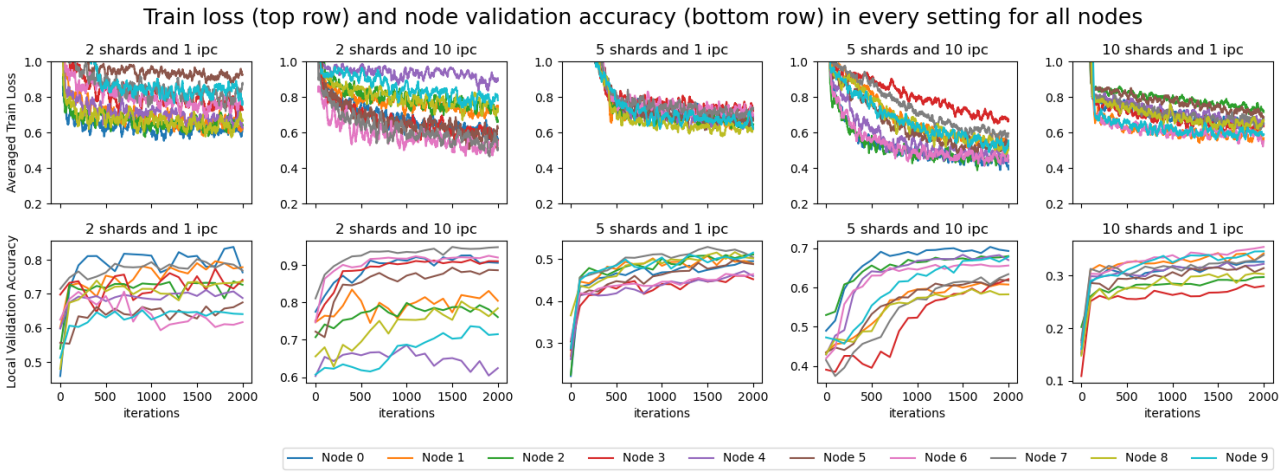

Figure 7. Training phase of data distillation. In the first row, we plot the averaged train loss for every setting across all nodes. The train loss is averaged using a sliding window of 30 epochs to reduce noise due to the randomly sampled starting epochs for the expert trajectories. In the second row, we plot the evolution of the validation accuracy on the respective validation sets during the training process for every setting.

and the nodes. In fact, communication is done in one single shot. On the other hand, the drawback of our method is that it comes with a very high computation cost, which may pose challenges considering the typically limited computation abilities of distributed devices.

V. Discussion and Limitations

In section IV-A, we evaluate the local performance of the distillation technique of Cazenavette et al. [1] on each node. In all studied configurations, the approach significantly outperforms the baseline method that we consider. While the obtained results are promising, we want to acknowledge one limitation of the distillation process.

As can be seen in Algorithm 1, the distillation technique employed in this study relies on several initialization parameters. In their work, Cazenavette et al. [1] conducted an ablation study to determine the optimal parameters on the complete CIFAR-10 dataset. However, due to computational constraints and limited resources, we were unable to perform a similar study on subsets of CIFAR-10. Instead, we re-used the initialization parameters determined by the authors and only made slight adjustments to them. This approach does not seem optimal as we detected instances of ill-training on certain nodes.

Figure 7 illustrates two key metrics for each setting and node: the averaged[1] training loss (top) and validation accuracy (bottom) across iterations. These metrics serve as indicators of proper training during the distillation process. For instance, let us consider the setting with 2 shards and 10 images per class. In this case, we clearly observe evidence of ill training in 5 out of the 10 participating nodes. For these nodes, the train loss does not properly decrease and the validation accuracy does not increase. We also notice similar instances of ill training, although to a lesser extent, in the settings with 2 shards and 1 image per class, as well as for 5 shards and 10 images per class.

Resolving the observed ill training behavior is challenging since, in all the examined setting, at least half of the participating nodes do not exhibit such issues. It remains to be explored whether it is possible to identify a set of hyper-parameters specific to each setting that would ensure proper training across all nodes. Alternatively, it might also be that an individual ablation study needs to be conducted for every subset of the dataset. In this case, the method would induce even more computational costs. Another possible explanation to this behaviour is simply that some base images are better suited for the distillation process. However, this explanation seems in contradiction with the setting with 5 shards and 1 image per class, for which the training behaviour is consistent across all nodes.

This limitation does not seem to have a direct negative impact the results we compute in table IV-A1. Indeed, the distillation process in the settings with 2 shards and 10 images per class, for which local training seems to be the most ill, does not yield worse results than other settings. Still, we believe that our method has not reached its full potential and that the results we computed can be further improved.

VI. Conclusion

In this work, we conducted an evaluation of a Federated Learning setting where distilled data instead of model weights are shared. We considered different data distributions among the nodes and different numbers of distilled images per class. First, we assessed the local performance of the distillation technique of Cazenavette et. al [1]. Then, we

[1]Train loss averaged using a sliding window of 30 epochs to reduce noise due to different starting epochs for expert trajectories.

trained a global model on the aggregated distilled datasets and evaluated its performance. While the outcomes obtained from our method show promise, they still fall short of competing with conventional FL techniques such as *FedAvg*. During the distillation process, we encountered instances of ill training behaviour which we think that if properly addressed, have the potential to improve the results we computed. The main advantage of our technique is that it provides a significant reduction in communication costs between nodes and servers. This opens up doors in term of data privacy, and distilled data exchange using encrypted channels can be considered. Further work is required to address the observed training issues and to evaluate the efficiency of the method across even more different data distributions. Moreover, it is necessary to compare the efficiency of our method against other existing distillation techniques.

ACKNOWLEDGEMENTS

I would like to thank Akash Dhasade and Rafael Pires for their continuous supervision throughout this project. Their expertise was of immense help to me and I am grateful for the time they have given me.

REFERENCES

[1] G. Cazenavette, T. Wang, A. Torralba, A. A. Efros, and J.-Y. Zhu, "Dataset distillation by matching training trajectories," 2022.

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2023.

[3] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *CoRR*, vol. abs/1602.05629, 2016. [Online]. Available: http://arxiv.org/abs/1602.05629

[4] R. Pathak and M. J. Wainwright, "Fedsplit: An algorithmic framework for fast federated optimization," 2020.

[5] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," 2021.

[6] A. Reisizadeh, F. Farnia, R. Pedarsani, and A. Jadbabaie, "Robust federated learning: The case of affine distribution shifts," 2020.

[7] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros, "Dataset distillation," 2020.

[8] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 2113–2122. [Online]. Available: https://proceedings.mlr.press/v37/maclaurin15.html

[9] B. Zhao, K. R. Mopuri, and H. Bilen, "Dataset condensation with gradient matching," 2021.

[10] K. Wang, B. Zhao, X. Peng, Z. Zhu, S. Yang, S. Wang, G. Huang, H. Bilen, X. Wang, and Y. You, "Cafe: Learning to condense dataset by aligning features," 2022.

[11] T. Nguyen, R. Novak, L. Xiao, and J. Lee, "Dataset distillation with infinitely wide convolutional networks," 2022.

[12] R. Yu, S. Liu, and X. Wang, "Dataset distillation: A comprehensive review," 2023.

[13] R. Song, D. Liu, D. Z. Chen, A. Festag, C. Trinitis, M. Schulz, and A. Knoll, "Federated learning via decentralized dataset distillation in resource-constrained edge environments," 2023.

[14] T. Nguyen, Z. Chen, and J. Lee, "Dataset meta-learning from kernel ridge-regression," 2021.

[15] A. Dhasade, A.-M. Kermarrec, R. Pires, R. Sharma, and M. Vujasinovic, "Decentralized learning made easy with DecentralizePy," in *Proceedings of the 3rd Workshop on Machine Learning and Systems*. ACM, may 2023. [Online]. Available: https://doi.org/10.11452F3578356.3592587

[16] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009.

[17] S. Gidaris and N. Komodakis, "Dynamic few-shot visual learning without forgetting," *CoRR*, vol. abs/1804.09458, 2018. [Online]. Available: http://arxiv.org/abs/1804.09458

[18] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," 2017.

[19] "Colaboratory," https://research.google.com/colaboratory/faq.html. Verified: 2022-03-24.

[20] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87 – 90.

[21] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.

[22] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.